**LUCRAREA DE LABORATOR Nr. 5**

*Tema:* **Tehnicile prelucrării tablourilor, utilizînd subprograme**

*Scopul lucrării:* însuşirea tehnicilor de algoritmizare şi programare cu subprograme în prelucrarea structurilor complexe în TP şi C.

*Obiectivele temei*

1. Aprofundarea cunoştinţelor în limbajul şi mediul TurboC şi perfecţionarea tehnicii de programare cu subprograme în prelucrarea structurilor complexe.
2. Însuşirea procedeelor de algoritmizare şi progamare prin descompunerea problemei în module autonome care, apoi, vor fi reprezentate prin secvenţe /fragmente de program pentru prelucrarea unor informaţii după principii comune cu apeluri multiple, analizând soluţiile stereotipe şi cele eficiente de introducere, afişare şi rearanjări ale tablourilor etc.
3. Însuşirea tehnicilor eficiente de parcurgere, căutare, schimbare şi ordonare a structurilor şi calculul conform cunoştinţelor obţinute din matematică în baza ciclurilor şi încorporate într-un program complex, alcătuit după principiile structurale şi modulare.

*Subiectele temei şi ordinea executării*

1. Studierea principiilor prelucrării (descrierii, declarării, formării, etc.) tablourilor.
2. Studierea metodei de descompunere top-down a programului pe module aparte care să fie apelate în diferite sectoare a programului, utilizînd subprograme.
3. Însuşirea tehnicilor moderne de elaborare a programelor complexe în C(cu subprograme proprii) în baza problemelor din lucr. de laborator nr.3,4.
4. Elaborarea testelor şi depanarea programului diverse moduri în mediul integrat C.
5. Compararea tehnicilor simple şi celor bazate pe principiile structurale şi modulare.

*Conţinutul raportului (vezi lucr. de laborator nr.1-3) şi s*uplimentar: Analiza erorilor admise pe parcursul efectuării lucrării şi eficienţa algoritmilor elaboraţi.

***Consideraţiile teoretice şi exemple***

***I. Subprograme în TP: PROCEDURI şi FUNCŢII***

După cum aţi făcut cunoştinţă în liceu şi la lucrarea de laborator nr.1, structura unui program simplu în Turbo Pascal cuprinde, de obicei, trei secţiuni, însă programele mai evoluate, care le vom numi complexe, în secţiunea de declaraţii , fiind alcatuită la rîndul ei din cinci subsectiuni, şi mai poate să conţină şi secţiunea de declarare a subprogramelor FUNCTION şi PROCEDURE, elaborate de programator, care reprezintă secvenţe de program subordonate, efectuate eventual în mod repetat, cu parametri diferiţi. Atunci secţiunea de instrucţiuni controlează execuţia programului, numindu-se de aceea şi program principal, şi face apel la eventualele subprograme FUNCTION şi PROCEDURE definite in secţiunea de declaraţii. Deci în programul complex, în afară de accesul la bibliotecile standarde ale TP pentru apeluri la subprogramele predefinite, se mai poate conţine una sau mai multe subprograme (module: proceduri şi funcţii) care sunt secvenţe de program subordonate şi se execută sub controlul lui. Ele sunt scrise o singură dată, însă pot fi apelate de un număr nelimitat după necesitate. Procedurile si funcţiile reprezintă aceeaşi structură în trei secţiuni (antet, secţiune de declaraţii şi secţiune de instrucţiuni) ca şi programul tradiţional scris la lucrările precedente.

***2. Subprograme în C: FUNCŢII***

**2.1. Funcţii si programarea structurata**

Programarea structurata este o metodă ce rezolva problema cu o strategie a programării mai eficientă si are următoarele principii:

1. Structurile de control trebuie sa fie cât se poate de simple;

2. Construcţia unui program trebuie sa fie descrisa top-down.

Descrierea top-down se refera la descompunerea problemei noastre in module (subprobleme). De obicei, aceste module sunt uşor de descris.

**2.1.1 Definirea şi apelarea funcţiilor în C.** În C noţiunea de funcţie este esenţială, deoarece asigură un mecanism de abstractizare a controlului: rezolvarea unei părţi a problemei poate fi încredinţată unei funcţii, moment în care suntem preocupaţi de ce face funcţia, fără a intra în detalii privind cum face funcţia anumite operaţii. Însăşi programul principal este o funcţie cu numele main(), iar programul C este reprezentat de o mulţime de definiri de variabile şi de funcţii.

Funcţiile pot fi clasificate în:

* • funcţii care întorc un rezultat;
* • funcţii care nu întorc nici un rezultat (similare procedurilor din Pascal).

Un program este compus din una sau mai multe funcţii, printre care si "main()" principala funcţie. Întotdeauna execuţia unui program începe cu "main()". Când o funcţie este apelata (sau invocata) atunci controlul programului este pasat funcţiei apelate. După ce aceasta îşi termina execuţia, atunci se pasează înapoi controlul către principala funcţie.

Codul sursă C care descrie ce face o funcţie se numeşte "definiţia funcţiei". Aceasta are următoarea formă generala:

**tip nume\_funcţie (lista\_parametri\_formali)**

**{**

**< declaraţia variabilelor locale, vizibile numai pentra această funcţie; >**

**< instrucţiuni executabile; >**

**return <rezultatul final>;**

**}**

Primul rând se numeşte "header-ul" (antetul) funcţiei, iar ceea ce este inclus intre acolade se numeşte corpul funcţiei. Daca in antet nu precizam parametrii, atunci se va scrie "void" (cuvânt rezervat pentru lista vida). Daca Funcţia nu întoarce nici o valoare, atunci se va scrie ca tip întors tot "void". Tipul returnat (întors) de funcţie este cel precizat in "return" (ce va fi îndată explicat). Parametrii din antetul funcţiei (lista\_parametri\_formali) sunt daţi printr-o lista cu argumente separate prin virgula. Aceste argumente sunt date de tipul argumentului urmat de un identificator ce aparţine acelui tip. Se mai spune ca acel identificator este "parametru formal".

**2.1.1.1 Exemplul definirii şi apelării funcţiilor în C, în stil vechi, când corpul funcţiei se amplasează înaintea lui main():**

**#include <…>**

**void tipareste\_mesaj(int k)**

**{ int i;**

**printf("Iti urez:\n");**

**for (i = 0; i < k; ++i)**

**printf(" O zi buna ! \n");**

**}**

**main()**

**{ int n;**

**printf("Dati un numar natural mic: ");**

**scanf("%d", &n);**

**tipareste\_mesaj(n); /** *apelul functiei tipareste\_mesaj /*

**}**

**Instrucţiunea "return"** Instrucţiunea "return" este folosita pentru doua scopuri. Când se executa o instrucţiune "return", controlul programului este pasat înapoi programului apelant. In plus, daca exista o expresie după acest "return", atunci se va returna valoarea acestei expresii. Instrucţiunea "return" poate avea formele:

**return;**

sau

**return expresie;**

Exemplul 2: Găsirea minimului a doi intregi.

**#include <…>**

**int min(int x, int y)**

**{**

**if (x < y) return x;**

**else return y**

**}**

**main()**

**{**

**int j, k, m;**

**printf("Dati doi intregi: ");**

**scanf("%d%d", &j, &k);**

**m = min(j, k); /** *apelul functiei* **min** */*

**printf("\n%d este minimul dintre %d si %d.\n", m, j, k);**

**}**

**2.1.1.2 Exemplul definirii funcţiilor în C prin *prototipurile funcţiilor*, în stil nou, când corpul funcţiei se amplasează după necesitate fără restricţii în afara lui main():**

In C, apelul unei funcţii poate apare înaintea declarării ei. Funcţia poate fi definita mai târziu in acelaşi fişier, sau in alt fişier sau dintr-o biblioteca standard. In ANSI C, prototipul funcţiei remediază problema punând la dispoziţie numărul si tipul argumentelor funcţiei. Prototipul specifica, de asemenea, si tipul returnat de funcţie.

Sintaxa prototipului unei funcţii este:

**tip nume\_funcţie (lista\_tipuri\_parametri\_formali);**

In lista de parametri putem specifica chiar si parametrul, dar acesta este optional. Daca Funcţia nu are parametri, atunci se foloseşte "void".

Exemplu: Reluam un exemplu precedent.

#include <…>

**main()**

**{**

**int n;**

**void tipareste\_mesaj(int);**

**printf("Dati un numar natural mic: ");**

**scanf("%d", &n);**

**tipareste\_mesaj(n);**

**}**

**void tipareste\_mesaj(k)**

**{**

**int i;**

**printf("Iti urez:\n");**

**for (i = 0; i < k; ++i)**

**printf(" O zi buna ! \n");**

**}**

Prototipul unei funcţii poate fi plasat in corpul altei funcţii, sau de regula, se scriu la începutul programelor după directivele #include <…> si #define.

**2.2. Descrierea metodei "top-down"** Presupunem ca avem de citit câţiva întregi si trebuie sa-i afişăm in ordine pe coloane (in capătul de sus al coloanelor trebuie sa scriem numele câmpului), sa le afişăm suma lor parţiala, minimul si maximul lor. Pentru scrierea unui program C ce face acest lucru, vom utiliza proiectarea (descrierea) "top-down".

Astfel, descompunem problema in următoarele module:

1. Un antet pentru problema data;

2. Scrierea câmpurilor;

3. Citirea si scrierea lor pe coloane.

Toţi aceşti trei paşi vor fi descrişi in câte o funcţie ce se apelează din "main()". Obţinem, un prim cod sursă:

#include <…>

**main()**

**{**

**void tipareste\_antet(void);**

**void scrie\_campurile(void);**

**void citeste\_scrie\_coloanele(void);**

**/** *apelurile funcţiilor:***/**

**tipareste\_antet();**

**scrie\_campurile();**

**citeste\_scrie\_coloanele();**

**}**

Aceasta reprezintă într-un mod foarte simplu descrierea "top-down". Daca o problema este prea grea, atunci o descompunem in module, si apoi le rezolvam pe acestea. Beneficiul suplimentar al acestei metode este claritatea sa.

**void tipareste\_antet(void)**

**{**

**printf("\n%s%s%s\n",**

**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",**

**"\* Calculul sumelor, minimului si maximului \*\n",**

**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**}**

Funcţia ce foloseşte la scrierea câmpurilor este la fel uşor de scris:

**void scrie\_campurile(void)**

**{**

**printf("%5s%12s%12s%12s%12s\n\n",**

**"Numar", "Articol", "Suma", "Minimul", "Maximul");**

**}**

Urmeaza apoi funcţia ce serveste la scrierea înregistrărilor referitoare la câmpurile discutate mai sus:

**void citeste\_scrie\_coloanele(void)**

**{**

**int contor = 0, articol, suma, minim, maxim;**

**int min(int, int), max(int, int);**

**if (scanf("%d", &articol) == 1)**

**{**

**++contor;**

**suma = minim = maxim = articol;**

**printf("%5d%12d%12d%12d%12d\n\n", contor, articol, suma, minim, maxim);**

**while (scanf("%d", &articol) == 1)**

**{**

**++contor;**

**suma += articol;**

**minim = min(articol, minim);**

**maxim = max(articol, maxim);**

**printf("%5d%12d%12d%12d%12d\n\n",**

**contor, articol, suma, minim, maxim);**

**}**

**}**

**else printf("Nici o data nu a fost citita.\n\n");**

**}**

Daca datele se introduc de la tastatura, atunci tabelul se va afişa "intrerupt" de citirile ce au loc de la tastatura. Astfel, se prefera citirea dintr-un fişier extern. Presupunem ca fişierul nostru executabil (asociat fişierului sursa scris in C) se numeşte "numere.exe" si am creat un fişier numit "fisier.int" ce contine urmatoarele numere:

19 23 -7 29 -11 17

Dând comanda

numere < fişier.int vom obtine un tabel ce contine toate datele dorite.

**Invocare si apel prin valoare** O funcţie este invocata prin scrierea numelui sau împreuna cu lista sa de argumente intre paranteze. De obicei, numărul si tipul acestor argumente se "potriveste" cu parametrii din lista de parametri prezenti in definitia funcţiei. Toate argumentele sunt apelate prin valoare ("call-by-value"). Asta inseamna ca fiecare argument este evaluat si valoarea sa este folosita ca valoare pentru parametrul formal corespunzator. De aceea, daca o variabila (argument) este folosita la transmiterea unei valori, atunci valoarea ei nu se schimba.

Exemplu:

#include <…>

**main()**

**{**

**int n=3, suma, calculeaza\_suma(int);**

**printf("%d\n", n);** /\* se va scrie 3 \*/

**suma = calculeaza\_suma(n);**

**printf("%d\n", n);** /\* se va scrie 3 \*/

**printf("%d\n", suma);** /\* se va scrie 6 \*/

**}**

**int calculeaza\_suma(int n)** /\* suma numerelor de la 1 la n \*/

**{**

**int suma = 0;**

**for ( ; n > 0; --n)** /\* n se schimba aici, dar nu si in main() \*/

**sum += n;**

**printf("%d\n", n);** /\* se va scrie 0 \*/

**return suma;**

**}**

Chiar daca **n** este trimis ca argument in funcţia "calculeaza\_suma()" si valoarea lui n se modifica in aceasta funcţie, valoarea sa din mediul apelant rămâne neschimbata. Vom vedea mai târziu cum se poate simula apelul prin adresa ("call-by-reference").

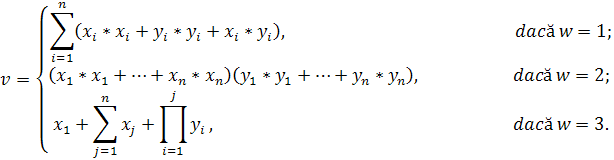
Deosebirea dintre "return" si "exit" Exista doua procedee de a returna o valoare.

**return expresie** si **exit(expresie)**

Daca se folosesc in "main()", atunci acestea sunt echivalente, dar in orice alta funcţie efectul lor este diferit (in ANSI C, Funcţia "main()" intoarce o valoare de tip int). Un apel al lui exit() in orice alta funcţie va implica terminarea execuţiei programului si returnarea valorii către mediul apelant (sistemul de operare sau mediul de programare C). Valoarea returnata se numeşte stare de iesire ("exit status"). Prin convenţie, starea de iesire zero indica terminare cu succes, pe cand iesire cu un numar diferit de zero indica o situatie anormala.

***Exemplu model: Scopul lucrarii :*** Însuşirea tehnicii top-down de algoritmizare şi programare cu funcţii şi în prelucrarea structurilor complexe în C/C++.

***Sarcina lucrarii :*** Sunt date tablourile X={xi}, i=1…n; Y={yi}, i=1…n şi valoarea w, în baza cărora să calculeze valorile lui v:



**Schema bloc:**

scanf(n,w)

i=0

i<n

scanf(x[i])

i++

w==1

v=first(x,y,n)

w==2

w==3

i=0

i<n

i++

v=pow(\*(a+i),2)+pow(\*(b+i),2)+\*(a+i)\*\*(b+i);

i=0

i<n

s1=s1+pow(\*(a+i),2)

s2=s2+pow(\*(b+i),2)

i++

j=0

j<n

s=1

i=0

i<j

s=s\*\*(b+i)

i++

v=v+\*(a+j)+s

j++

v=second(x,y,n)

v=third(x,y,n)

**Listingul :**

#include<stdio.h>

#include<conio.h>

#include<math.h>

**int first(int x[],int y[],int n)**

{

int i,v,\*a,\*b;

a=x;b=y;

for(i=0;i<n;i++)

v=pow(\*(a+i),2)+pow(\*(b+i),2)+\*(a+i)\*\*(b+i);

return v;

}

**int second(int x[],int y[],int n)**

{

int i,v,\*a,\*b,s1,s2;

a=x;b=y;s1=0;s2=0;

for(i=0;i<n;i++)

{

s1=s1+pow(\*(a+i),2);

s2=s2+pow(\*(b+i),2);

}

return s1\*s2;

}

**int third(int x[],int y[],int n)**

{

int j,i,v,\*a,\*b,s;

a=x;b=y;v=x[1];

for(j=0;j<n;j++)

{

s=1;

for(i=0;i<j;i++)

s=s\*\*(b+i);

v=v+\*(a+i)+s;

}

return v;

}

main()

{

clrscr();

int i,j,n,x[50],y[50],v,w;

printf("Introduceti marimea masivelor:");

scanf("%d",&n);

printf("\nIntroduceti valorile masivului x:");

for(i=0;i<n;i++)

scanf("%d",&x[i]);

printf("\nIntroduceti valorile masivului y:");

for(i=0;i<n;i++)

scanf("%d",&y[i]);

printf("\nIntroduceti w:");

scanf("%d",&w);

**if(w==1) v=first(x,y,n);**

**if(w==2) v=second(x,y,n);**

**if(w==3) v=third(x,y,n);**

printf("\nRezultatul este: %d",v);

getch();}

Exercitii propuse spre implementare

1. Folosind funcţiile "rand()", "min(,)" si "max(,)", sa se genereze **n** numere naturale si sa se afiseze minimul si maximul dintre acestea.

2. (Jocul cap-pajura, simulare Monte-Carlo) Presupunem ca dispunem de o moneda ideala (nemasluita). Doi jucatori arunca cu moneda dupa urmatoarele reguli:

1.a. Se fac un numar total de n aruncari;

1.b. Primul jucator arunca moneda si celalalt spune "cap" sau "pajura";

1.c. Daca acesta "ghiceste" ce va pica moneda, atunci se inverseaza jucatorii (adica arunca al doilea si primul incearca sa ghiceasca);

1.d. La sfarsit, trebuie afisat scorul (si procentul de castig al fiecaruia).

3. (Conjectura lui Goldbach)

Orice numar par mai mare decat 2 se poate scrie ca suma a doua numere prime. Scrieti un program C care verifica aceasta conjectura pentru numere situate intre m si n. De exemplu, daca m=700 si n=1100, atunci afisati:

700 = 17 + 683

702 = 11 + 691

704 = 3 + 701

...

1098 = 5 + 1093

1100 = 3 + 1097

Generalizare: Scrieti toate combinatiile posibile de adunare a doua numere prime egal cu un numar dat.

## Bibliografie

1. D. Lucanu: Bazele proiectarii programelor si algoritmilor, Universitatea “A.I.Cuza” Iasi, 1996
2. L. Livovschi, H. Georgescu: Sinteza si analiza algoritmilor, Ed. Stiintifica si enciclopedica, 1986
3. O. Catrina, I. Cojocaru, Turbo C++, ed. Teora 1993
4. V. Petrovici, Florin Goicea, Programarea in limbajul C. Eed. Teora 1999
5. Liviu Negrescu, ,,Limbajul C” ,volumul I\_partea I-a si partea II-a7 7ditura MicroInformatica, Cluj-napoca 2001
6. Б.Керниган, Д.Ритчи. Язык программирования Си. Санкт-Петербург, 2001,,, Brian Kernighan, Dennis Ritchie *” The C Programming Language” este în l. română format electronic*
7. Indrumar de lucrări de laborator: “Informatica”. /A.Popescu, Ş. Marin / Chisinau. UTM,2003.
8. Vlad Caprariu ”Ghid de utilizare Turbo C” Cluj - Napoca 1993.
9. Cristea Valentin. Tehnici de programare. Ed.: Bucur., Teora, 1993. /681.3; T29/
10. Odagescu Ioan, Copos Cristina s.a. Metode si Tehnici de programare./enunturi, solutii, probleme propuse/ Ed.:Bucur.: INTACT, 1994 /681.3; O23/
11. Tudor Bălănescu. Corectudinea agoritmilor.Bucur.:Ed. Tehn.1995